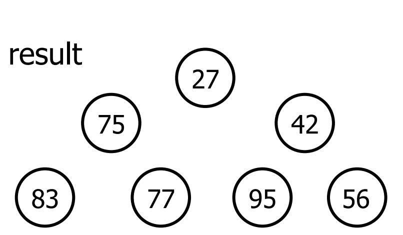
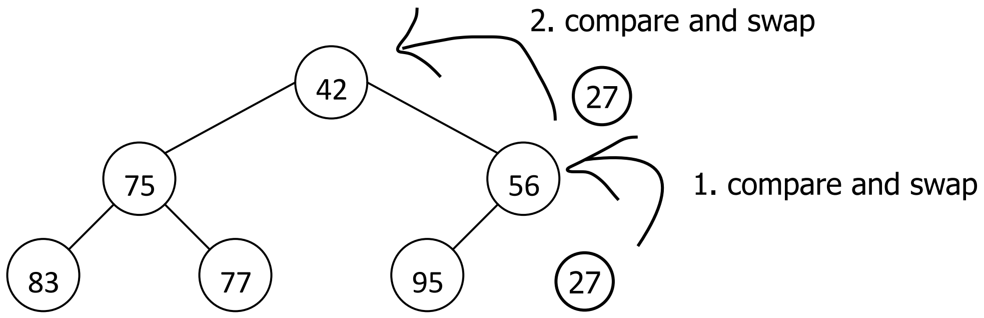
CS526 O2

Homework Assignment 4

Problem 1 (10 points). Consider the following heap, which shows integer keys in the nodes:

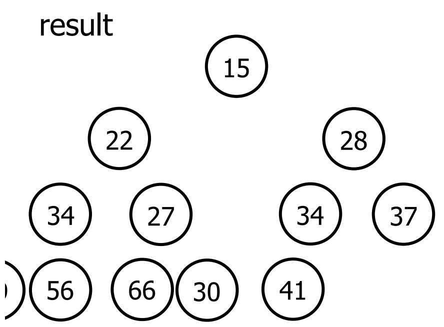


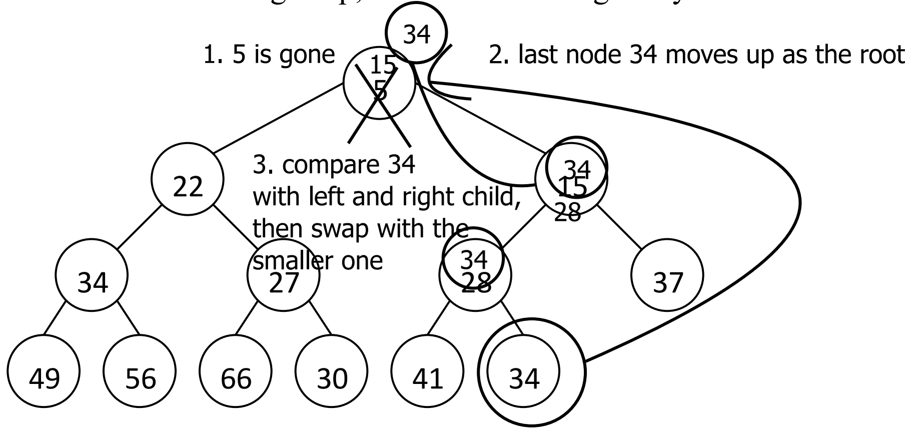


Show the resulting tree if you add an entry with key = 27 to the above tree? You need to describe, step by step, how the resulting tree is generated.

First, 27 will be added to the end, then we reorganize the heap to maintain heap-order property which the parent key should always be smaller or equal to the child key. When adding an element to the heap, it uses up-heap bubbling that bubbling all the way back to the root by finding last empty location, swap based on the priority with the parent, until we get the point that the child is no longer smaller than parent. Therefore, 27 compares with its parent 56, since 27 is smaller, so it swaps with 56, then it compares with the root 42, it is smaller than 42, so it swaps with 42. Now, 27 will become the new root of the heap.

Problem 2 (10 points). Consider the following heap, which shows integer keys in the nodes:




1. 5 is gone
2. last node 34 moves up as the root
3. compare 34 with left and right child, then swap with the smaller one
34
15
5
22
34
15
28
34
27
34
28
37
49
56
66
30
41
34

Suppose that you execute the *removeMin*( ) operation on the above tree. Show the resulting tree. You need to describe, step by step, how the resulting tree is generated.

<5, 22, 15, 34, 27, 28, 37, 49, 56, 66, 30, 41, 34> - initial, then we execute removeMin() where we remove the root, 5 is removed, last entry 34 moves up to be the new root and performs down-heap bubbling, where each time we compare the current node with both left and right child, and replace it with the minimum child. As 34 moves up, it compares 22 and 15, 15 is smaller so 15 moves up, then 34 compares with its current children 28 and 37, where 28 is smaller so 28 moves up, then 34 is smaller then 41 so it can stays with where it is now. The result should be <15, 22, 28, 34, 27, 34, 37, 49, 56, 66, 30, 41> now.

Problem 3 (10 points). This problem is about the chaining method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

Sequence of keys to be inserted: <5, 8, 44, 23, 12, 20, 35, 32, 14, 16>

0	1	2	3	4	5	6	7	8	9	10
44	23	35	14		5			8	20	32
	12				16					

5 mod 11 = 5 - first in address 5
8 mod 11 = 8
44 mod 11 = 0
23 mod 11 = 1 - first in address
12 mod 11 = 1 - placed below 23
20 mod 11 = 9
35 mod 11 = 2
32 mod 11 = 10
14 mod 11 = 3
16 mod 11 = 5 - placed below 5

Problem 4 (10 points). This problem is about linear probing method we discussed in the class. Consider a hash table of size N = 11. Suppose that you insert the following sequence of keys to an initially empty hash table. Show, step by step, the content of the hash table.

Sequence of keys to be inserted: <5, 8, 44, 23, 12, 20, 35, 32, 14, 16>

0	1	2	3	4	5	6	7	8	9	10
44	23	12	35	14	5	16		8	20	32

5 mod 11 = 5
8 mod 11 = 8
44 mod 11 = 0
23 mod 11 = 1
12 mod 11 = 1 - move to index
20 mod 11 = 9
35 mod 11 = 2 - move to index
32 mod 11 = 10
14 mod 11 = 3 - move to index
16 mod 11 = 5 - move to index

Problem 5 (10 points). Suppose that your hash function resolves collisions using open addressing with double hashing, which we discussed in the class. The double hashing method uses two hash functions *h* and *h*'.

Assume that the table size $N = 13$, $h(k) = k \bmod 13$, $h'(k) = 1 + (k \bmod 11)$, and the current content of the hash table is:

0	1	2	3	4	5	6	7	8	9	10	11	12
	16	2	29		18			21	48	15		

If you insert $k = 16$ to this hash table, where will it be placed in the hash table? You must describe, step by step, how the location of the key is determined.

h(k) = k mod 13
h(16) = 16 mod 13 = 3

h′(k) = 1 + (k mod 11)
h′(16) = 1 + (16 mod 11) = 6

h(k, i) = (h(k) + i*h′(k)) mod N
i = 0;
[3+0(6)]mod13=3 => index occupied
i = 1;
[3+1(6)]mod13=9 => index occupied
i = 2;
[3+2(6)]mod13=2 => index occupied
i = 3;
[3+3(6)]mod13=8 => index occupied
i = 4;
[3+4(6)]mod13=1 => probe

Please see Hw4.p6.pdf for the Answer to Discussion/observation of Problem 6

Problem 6 (50 points). The goal of this problem is to give students an opportunity to observe differences among three data structures in Java – Java's *HashMap*, *ArrayList*, *LinkedList* – in terms of insertion time and search time.

Students are required to write a program that implements the following pseudocode:

```
create a HashMap instance myMap
create an ArrayList instance myArrayList
create a LinkedList instanc myLinkedList

Repeat the following 10 times and calculate average total insertion time and average total
search time for each data structure

        generate 100,000 distinct random integers in the range [1, 1,000,000] and store them in
        the array of integers insertKeys[ ]

        // begin with empty myHap, myArrayList, and myLinkedList each time

        // Insert keys one at a time but measure only the total time (not individual insert  //
        time)
        // Use put method for HashMap, e.g., myMap.put(insertKeys[i], i)
        // Use add method for ArrayList and LinkedList

        insert all keys in insertKeys [ ] into myMap and measure the total insert time
        insert all keys in insertKeys [ ] into  myArrayList and measure the total insert time
        insert all keys in insertKeys [ ] into myLinkedList and measure the total insert time

        generate 100,000 distinct random integers in the range [1, 2,000,000] and store them in
        the array searchKeys[ ].

        // Search keys one at a time but measure only total time (not individual search //
        time)
        // Use containsKey method for HashMap
        // Use contains method for ArrayList and Linked List

        search myMap for all keys in searchKeys[ ] and measure the total search time search
        myArrayList for all keys in searchKeys[ ] and measure the total search time  search
        myLinkedList for all keys in searchKeys[ ] and measure the total search time
```

Print your output on the screen using the following format:

```
        Number of keys = 100000

        HashMap average total insert time = xxxxx
        ArrayList average total insert time = xxxxx
        LinkedList average total insert time = xxxxx

        HashMap average total search time = xxxxx
        ArrayList average total search time = xxxxx
        LinkedList average total search time = xxxxx
```

You can generate *n* random integers between 1 and N in the following way:

```
Random r = new Random(System.currentTimeMillis() );
for i = 0 to n – 1
    a[i] = r.nextInt(N) + 1
```

When you generate random numbers, it is a good practice to reset the seed. When you first create an instance of the Random class, you can pass a seed as an argument, as shown below:

```
Random r = new Random(System.currentTimeMillis());
```

You can pass any long integer as an argument. The above example uses the current time as a seed.

Later, when you want to generate another sequence of random numbers using the same Random instance, you can reset the seed as follows:

```
r.setSeed(System.currentTimeMillis());
```

You can also use the *Math.random*( ) method. Refer to a Java tutorial or reference manual on how to use this method.

We cannot accurately measure the execution time of a code segment. However, we can estimate it by measuring an elapsed time, as shown below:

```
long startTime, endTime, elapsedTime; startTime
= System.currentTimeMillis();
// code segment
endTime = System.currentTimeMillis(); elapsedTime
= endTime - startTime;
```

We can use the *elapsedTime* as an estimate of the execution time of the code segment. Note that if the elapsed time is similar for the three data structures, you may need a more precise measure using System.nanoTime() or something similar.

Name the program *Hw4_P6.java*.

Deliverable

You need to submit the following files:

- *Hw4.p1_p5pdf*: This file must include:
- Answers to problems 1 through 5.

- Discussion/observation of Problem 6: This part must include what you observed and learned from this experiment and it must be "substantive."
- *Hw4_p6.java*
- Other files, if any.

Combine all files into a single archive file and name it *LastName_FirstName_hw4.EXT*, where *EXT* is an appropriate archive file extension, such as *zip* or *rar*.

Grading

Problem 1 through Problem 5:

- For each problem, up to 6 points will be deducted if your answer is wrong.

Problem 6:

- There is no one correct output. As far as your output is consistent with generally expected output, no point will be deducted. Otherwise, up to 20 points will be deducted.
- If your conclusion/observation/discussion is not substantive, points will be deducted up to 5 points.
- If there are no sufficient inline comments in your program, points will be deducted up to 5 points.